Hideo Okawara's
Mixed Signal Lecture Series

DSP-Based Testing – Fundamentals 14
FIR Filter

Verigy Japan

June 2009

Preface to the Series

ADC and DAC are the most typical mixed signal devices. In mixed signal testing, analog stimulus signal is generated by an arbitrary waveform generator (AWG) which employs a D/A converter inside, and analog signal is measured by a digitizer or a sampler which employs an A/D converter inside. The stimulus signal is created with mathematical method, and the measured signal is processed with mathematical method, extracting various parameters. It is based on digital signal processing (DSP) so that our test methodologies are often called DSP-based testing.

Test/application engineers in the mixed signal field should have thorough knowledge about DSP-based testing. FFT (Fast Fourier Transform) is the most powerful tool here. This corner will deliver a series of fundamental knowledge of DSP-based testing, especially FFT and its related topics. It will help test/application engineers comprehend what the DSP-based testing is and assorted techniques.

Editor's Note

For other articles in this series, please visit the Verigy web site at www.verigy.com/go/gosemi.

FIR Filter

FIR or finite impulse response is a time domain waveform which can be utilized for digital filtering with convolution mathematics. Filtering by the FFT-IFFT method discussed in the previous article has a restriction that the waveform length should be 2^n points. Convolution does not have such restriction so that, if the number of data is not 2^n, FIR filtering can be useful. In this article, you will see how to create a FIR simply and how to play filtering. More over FIR can be generated with utilizing IFFT. So this is one of the practical applications of IFFT. In order to check the frequency response, a multi-tone signal is used. You can generate a time-domain waveform by using IFFT in the frequency domain. It is described as an example too. So in this article you will see two practical applications of IFFT.

1. Filtering

As learned in FFT-IFFT method, filtering is to remove some part of the frequency components from the signal frequency spectrum. Mathematically it can be realized with multiplication of a desired frequency characteristics $H(f)$ to the signal frequency spectrum $X(f)$. In Figure 1, the filtering can be described as $H(f) \cdot X(f)$.

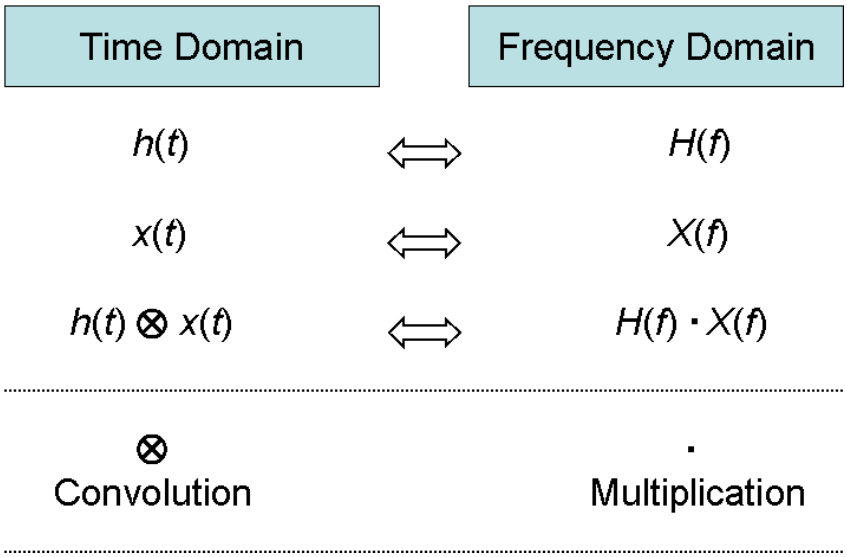

Figure 1: Time Domain vs. Frequency Domain

In the Fourier transform relationship, multiplication in the frequency domain corresponds to convolution in the time domain. So filtering in the time domain can be realized with convolution of an impulse response $h(t)$ to an signal waveform $x(t)$.

2. How to Create an FIR

First off, you need to design the frequency characteristics of your desired filter. Let's exercise to design a low pass filter (LPF) with defining following parameters.

Sampling Frequency (Digitizer/Sampler/ADC):	*Fs*
Number of Sampling Points:	*N*
LPF Cut-off Frequency:	*Fco*
FIR Length:	*Nfir*

Figure 2 shows the desired frequency response in the frequency domain. The gain is 1.0 from DC to the cut-off frequency *Fco*. The gain from *Fco* to the Nyquist frequency *Fs*/2 is set 0.0 in the left page. The FIR will be created with utilizing IFFT so that you have to prepare the full-page spectrum as Figure 2. The right page must be complex conjugate; however, actually the imaginary part is zero so that the right page can be created symmetrically to the left page.

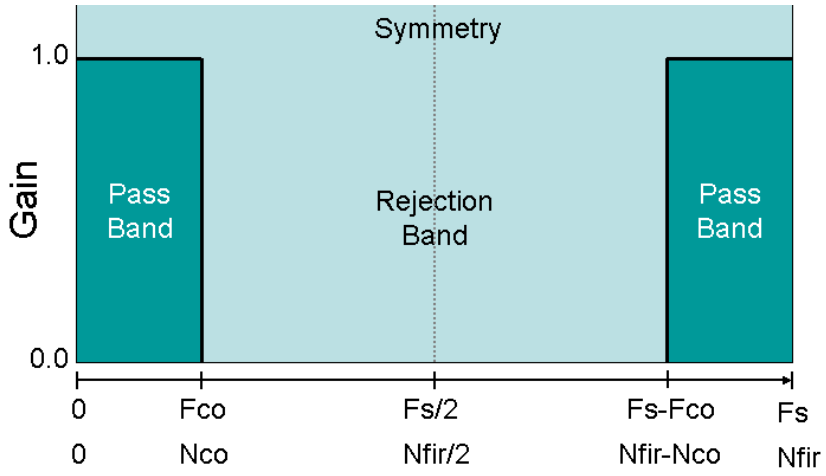

Figure 2: Design of LPF Frequency Characteristics

The coding for the response in Figure 2 looks as follows;

```
01:     INT             i, j, Nfir, Nco, Nsp;
02:     DOUBLE          dTemp;
03:     ARRAY_D         dFIR;
04:     ARRAY_COMPLEX   CSp,CFIR;
05:
06:     Nfir=128;                       // FIR length as you like
07:     Nco=13;                         // Cut-off Frequency=Fs*(Nco/Nfir)
08:     CSp.resize(Nfir);               // To create a full-page spectrum
09:
10:     CSp[0].real()=1.0;
11:     CSp[0].imag()=0.0;
12:     for (i=1;i<=Nco;i++) {                  // Pass-band
13:        CSp[i].real()=1.0;                     // Left Page (Gain=1.0)
14:        CSp[i].imag()=0.0;
15:        CSp[Nfir-i].real()=1.0;                // Right Page (Gain=1.0)
16:        CSp[Nfir-i].imag()=0.0;
17:     }
18:     for (i=Nco+1;i<=(Nfir-Nco-1);i++) {     // Rejection band
19:        CSp[i].real()=0.0;                     // Gain=0.0
20:        CSp[i].imag()=0.0;
21:     }
22:     DSP_IFFT(CSp,CFIR);
23:     dFIR.resize(Nfir);
24:     dFIR=CFIR.getReal();                    // Pick up the real part
25:                                             // Primitive FIR
```

List 1: FIR Generation for LPF

In List 1, the FIR length (*Nfir*) is defined 128. The size is up to you, but it should be 2^n for IFFT. The brick-wall boundary (*Nco*) is defined 13 here so that the actual cut-off frequency is settled as 13/128 of the sampling frequency (*Fs*).

Performing IFFT at Line 22 in List 1, a primitive impulse response is generated as shown in Figure 3, where the created FIR is actually split half and distributed separately in the left and right pages.

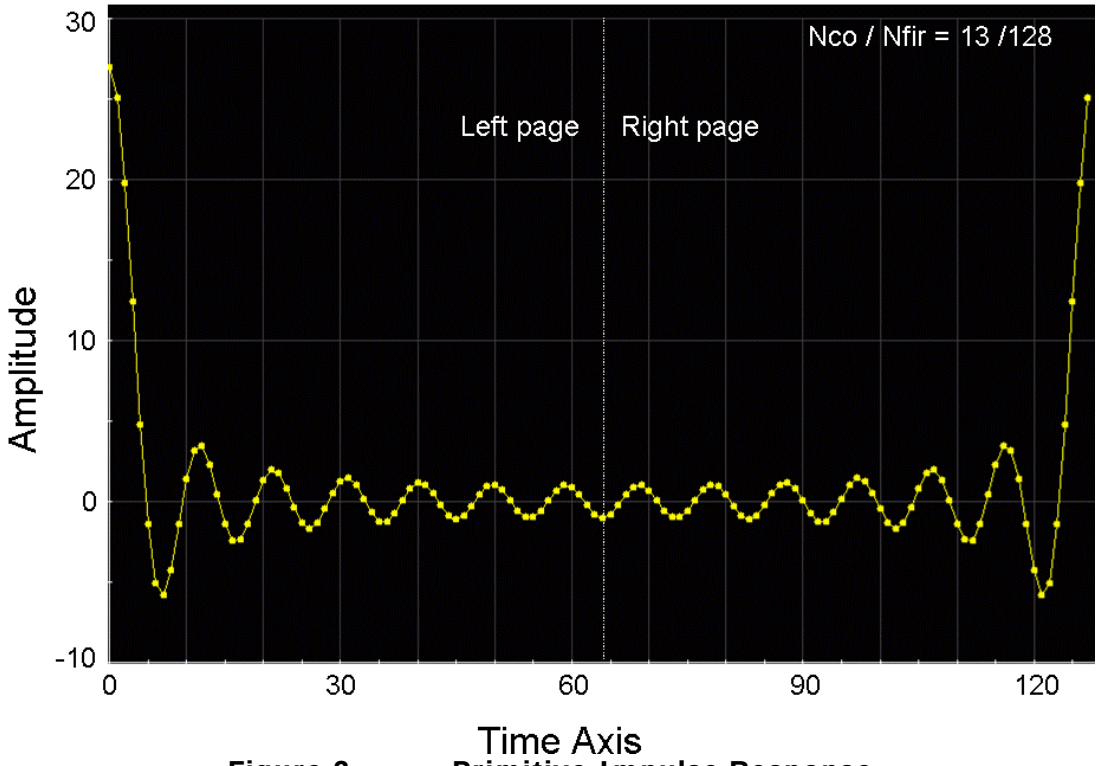

Figure 3: Primitive Impulse Response

To reconstruct a correct impulse shape, the left and the right pages should be exchanged with each other by the routine in List 2.

```
26:     Nsp=Nfir/2;
27:     for (i=0;i<Nsp;i++) {           // Exchange left and right pages
28:        dTemp=dFIR[i];
29:        dFIR[i]=dFIR[Nsp+i];
30:        dFIR[Nsp+i]=dTemp;
31:     }
32:
```

List 2: Exchange Left and Right Pages

Then true FIR is reconstructed as shown in Figure 4.

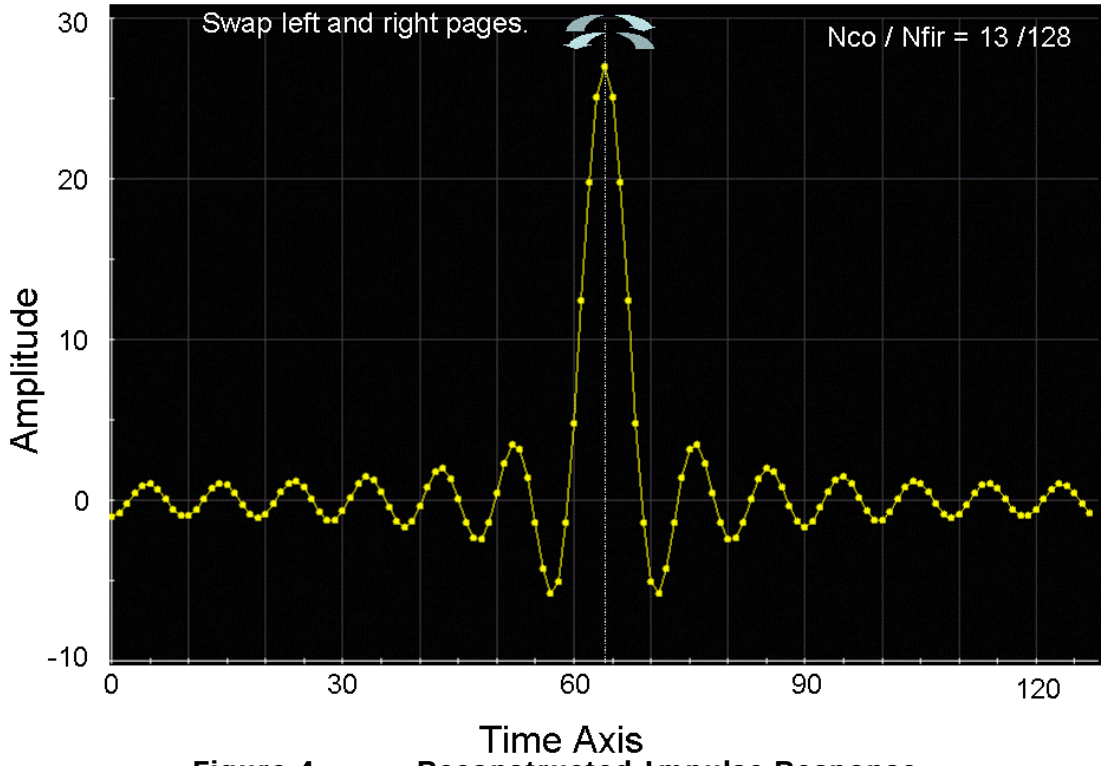

Figure 4: Reconstructed Impulse Response

Since the original frequency characteristics in Figure 2 is brick-shape, the impulse response actually becomes infinitely oscillating "sinX/X" (SINC) curve. FIR or "Finite" impulse response comes from the fact that a finite period is taken out of the infinitely continuous curve.

This is a primitive FIR so that you should normalize the impulse response divided by the FIR length. Otherwise the output signal amplitude would get impacted. The scaling routine is quite simple as follows.

```
32:
33:     DSP_MUL_SCL(1.0/(DOUBLE)Nfir,dFIR,dFIR);   // Scaling
34:
```

List 3: Normalization of Impulse Response

The reconstructed and normalized FIR looks as Figure 5.

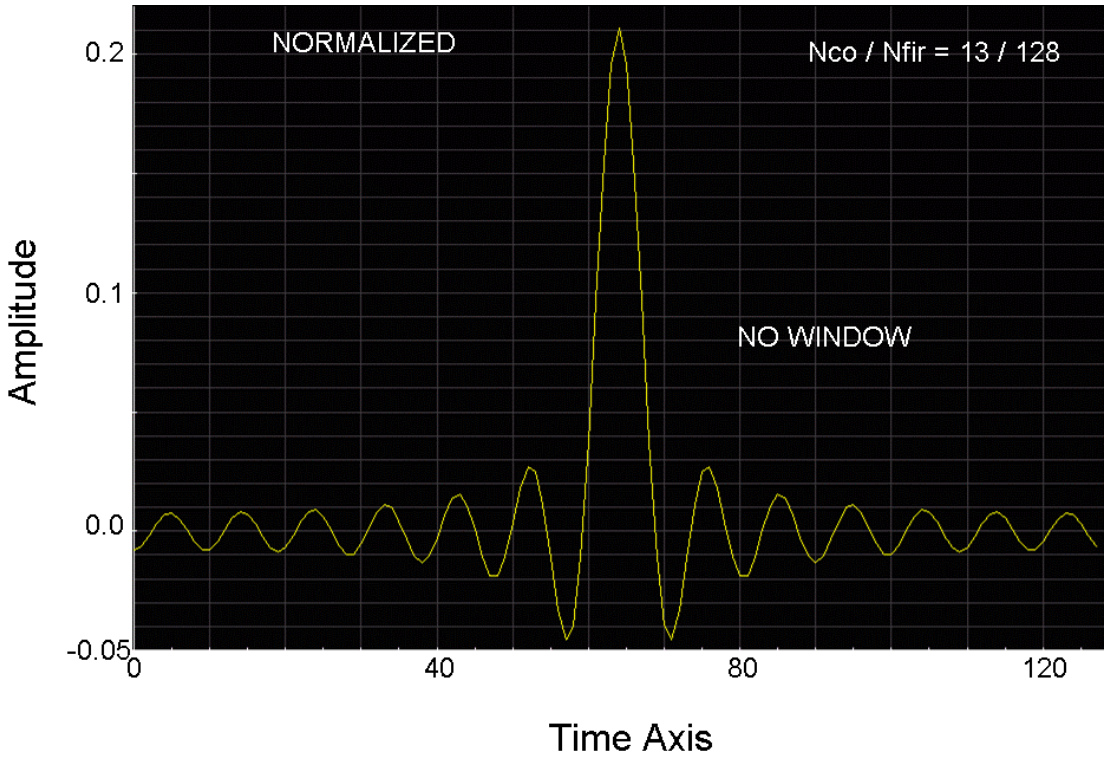

Figure 5: Normalized FIR

3. Filtering Operation with FIR

Once you create an FIR array, a convolution routine is provided as a DSP API in the V93000 test system so that the filtering operation is simply performed as follows;

```
35:     ARRAY_D      dVsmp;
36:
37:     dVsmp=DGT("AOUT").getWaveform();          // Upload digitizer data
38:     DSP_CONVOL(dFIR,dVsmp,dWave,ON);          // Filtering
39:
```

List 4: Convolution or FIR Digital Filtering

In the API DSP_CONVOL(), basically the sizes of "dFIR" and "dVsmp" are not necessarily restricted to 2^n. However, the FIR array "dFIR" is generated by using the IFFT method so that its size should be equal to 2^n here. The input waveform array "dVsmp" is free from 2^n. If "dVsmp" contains an integer number of cycles of the test signal waveform, the 4^{th} parameter is set "ON" for circular operation. If not, you may need to set "OFF" and then you would have garbage data in the both ends of the output array "dWave" by the half length of FIR as Figure 6(a) shows. If you want to get clean result in the whole period in this case, measure and capture longer points in the input array "dVsmp" by the length of FIR as Figure 6(b). After convolution, cut off the garbage data at both ends by the half length of FIR each.

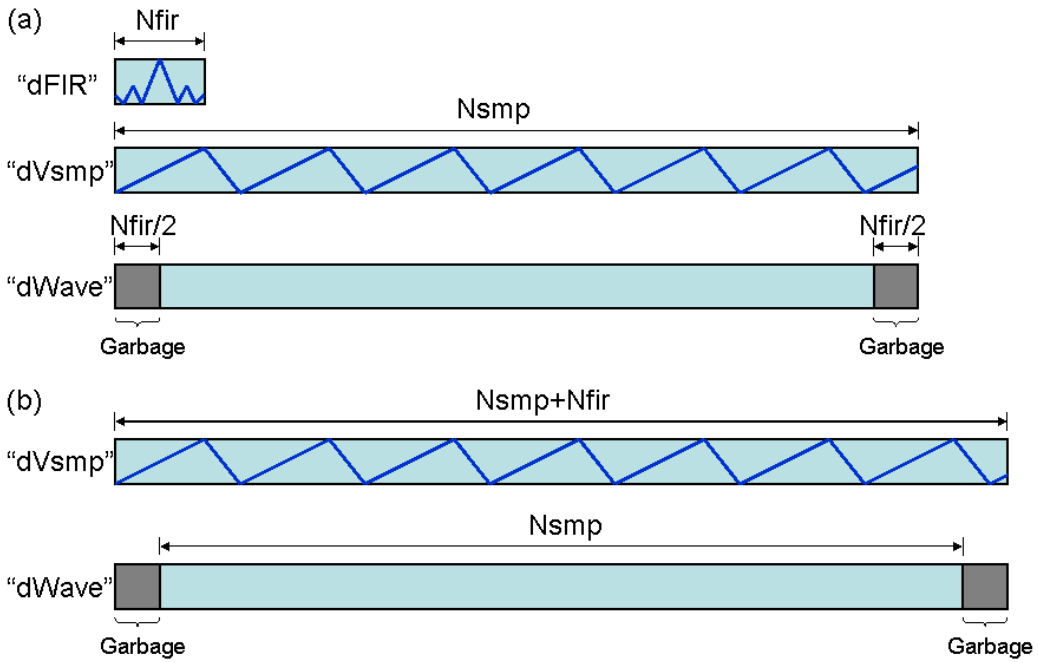

Figure 6: DSP_CONVOL(, , ,OFF)

4. Check Frequency Response

Let's look at the performance of the primitive FIR filter. In order to see the filter frequency response, a multi-tone signal is very efficient. Let's create a full-range (DC to the Nyquist) multi-tone as follows. In terms of the hyperbolic phase control in multi-tone generation, refer to the previous article "DSP-Based Testing -- Fundamentals 4."

```
40:     INT       Nsmp,Ns;
41:     DOUBLE    dP,dQ;
42:     ARRAY_D   dWave;
43:
44:     Nsmp=8192;                  //  # of data points here
45:     Ns=Nsmp/2;
46:     dWave.resize(Nsmp);         //  Test Signal Container
47:     dWave.init(1.0);            //  DC 1V
48:     for (j=1;j<Ns;j++) {        //  Each AC 1V with hyperbolic phase
49:        dP=2.0*M_PI*j/Nsmp;
50:        dQ=M_PI/(Ns-1)*(1-j*j);
51:        for (i=0;i<Nsmp;i++) dWave[i]=dWave[i]+cos(dP*i+dQ);
52:     }
53:
```

List 5: Test Signal Multi-tone Generation (Time Domain Method)

The test signal generated by the source code in List 5 contains 1V components from DC to the bin #(*Ns*-1). Therefore the convolution of the FIR to this test multi-tone directly shows the frequency characteristics of the FIR filter.

By the way, the multi-tone in List 5 is directly generated in the time domain. Actually this method takes long time to execute. There is another elegant way to generate such a waveform, and it is actually much faster than that. See List 6. The programming is done in the frequency domain, and it is another application of IFFT.

```
54:     ARRAY_COMPLEX   CSpA,CWave;
55:
56:     CSpA.resize(Nsmp);                // Spectrum Container
57:     CSpA[0].real()=1.0;
58:     CSpA[0].imag()=0.0;               // DC 1.0V
59:     for (j=1;j<Ns;j++) {
60:        dQ=M_PI/(Ns-1)*(1-j*j);      // Hyperbolic Phase Offset
61:        CSpA[j].real()=0.5*cos(dQ);          // Each AC 1.0 V
62:        CSpA[j].imag()=0.5*sin(dQ);          //(Half length)
63:        CSpA[Nsmp-j].real()= CSpA[j].real(); // Complex
64:        CSpA[Nsmp-j].imag()=-CSpA[j].imag(); // Conjugate
65:     }
66:
67:     DSP_IFFT(CSpA,CWave);
68:     dWave.resize(Nsmp);
69:     dWave=CWave.getReal();                  // Take Real Part
70:
```

List 6: Test Signal Multi-tone Generation (Frequency Domain Method)

Probably the frequency domain method (List 6) would be several tens of times faster than the time domain method (List 5). Figure 6 shows the waveform generated by the source code in List 6.

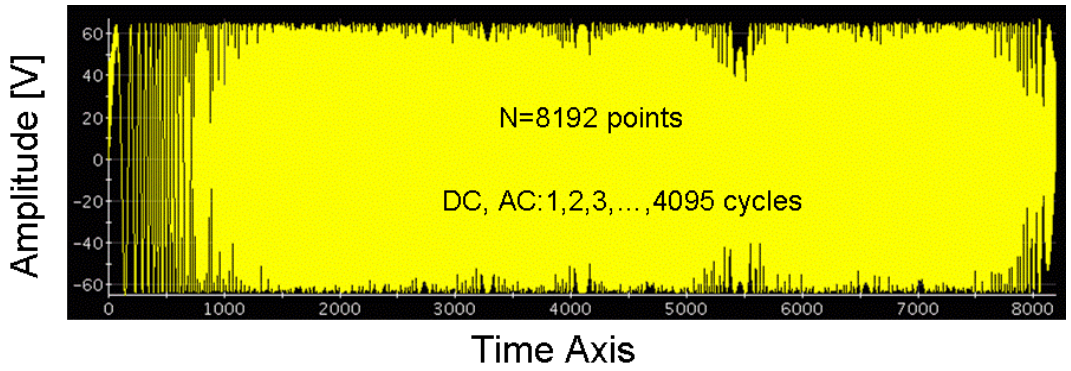

Figure 6: Test Signal Waveform (Multi-tone)

The waveform in Figure 6 contains entire components from DC to 4095 cycles. This input waveform is convoluted with the FIR in Figure 5. The output waveform is processed with FFT, and the frequency response is shown in Figure 7. This is the frequency characteristics of the LPF whose FIR is shown in Figure 5. The programmed cut off frequency is 8192*13/128=832. Figure 7 certainly shows that the roll-off shoulder is located around 800. However, the pass-band ripple is significant. The first side-lobe in the rejection is only -17dB so that the total rejection trend is not excellent. This is a poor LPF. The unsatisfactory performance is caused by the fact that the infinitely continuous impulse response is suddenly broken off at a finite period.

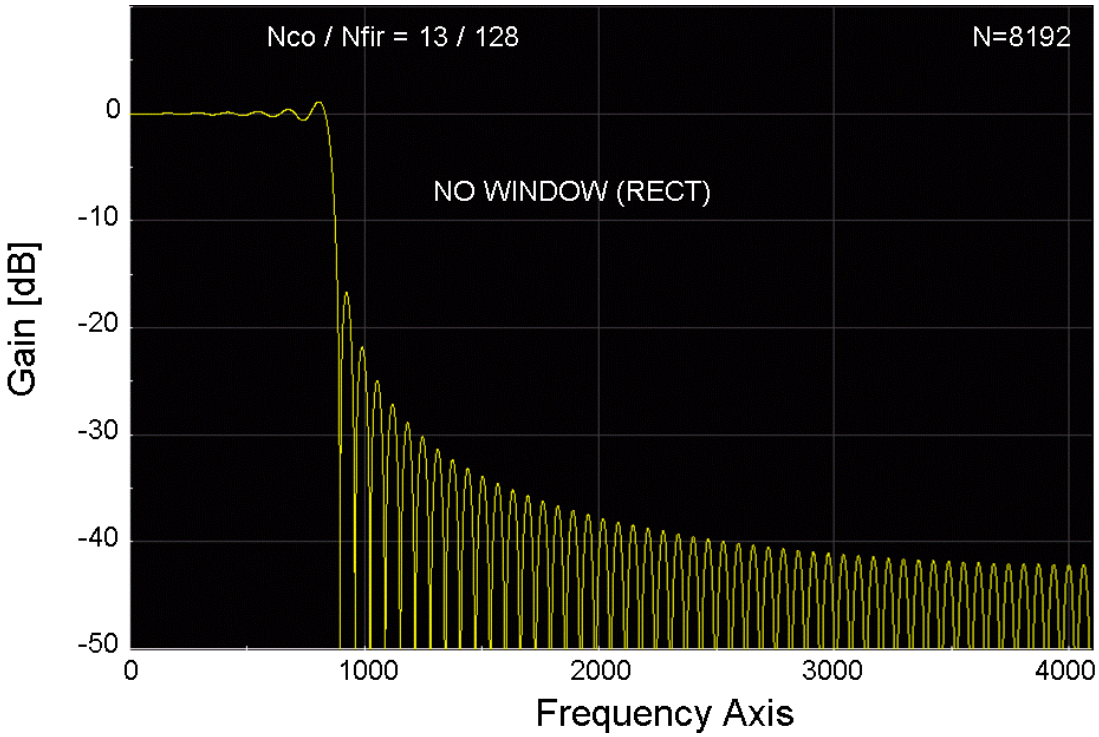

Figure 7: Frequency Response of FIR of 13 / 128

5. Practical FIR Generation

In order to improve the poor performance of the primitive FIR filter, windowing is very effective. The point is that the tails of FIR are asymptotically suppressed to zero. There are several window functions available. Choose window functions whose terminals converge zero. Here Hanning, Blackman and Blackman-Harris windows are selected. Three windows are described as follows.

Hanning: $$w(k) = \frac{1}{2}\left(1 - \cos\left(\frac{2\pi k}{N}\right)\right) \quad (1)$$

Blackman: $$w(k) = \frac{1-\alpha}{2} - \frac{1}{2}\cos\left(\frac{2\pi k}{N}\right) + \frac{\alpha}{2}\cos\left(\frac{4\pi k}{N}\right) \quad (2)$$

(α=0.16)

Blackman-Harris: $$w(k) = a_o - a_1\cos\left(\frac{2\pi k}{N}\right) + a_2\cos\left(\frac{4\pi k}{N}\right) - a_3\cos\left(\frac{6\pi k}{N}\right) \quad (3)$$

(a_o=0.35875, a_1=0.48829, a_2=0.14128, a_3=0.01168)

The shapes of these windows are shown in Figure 8.

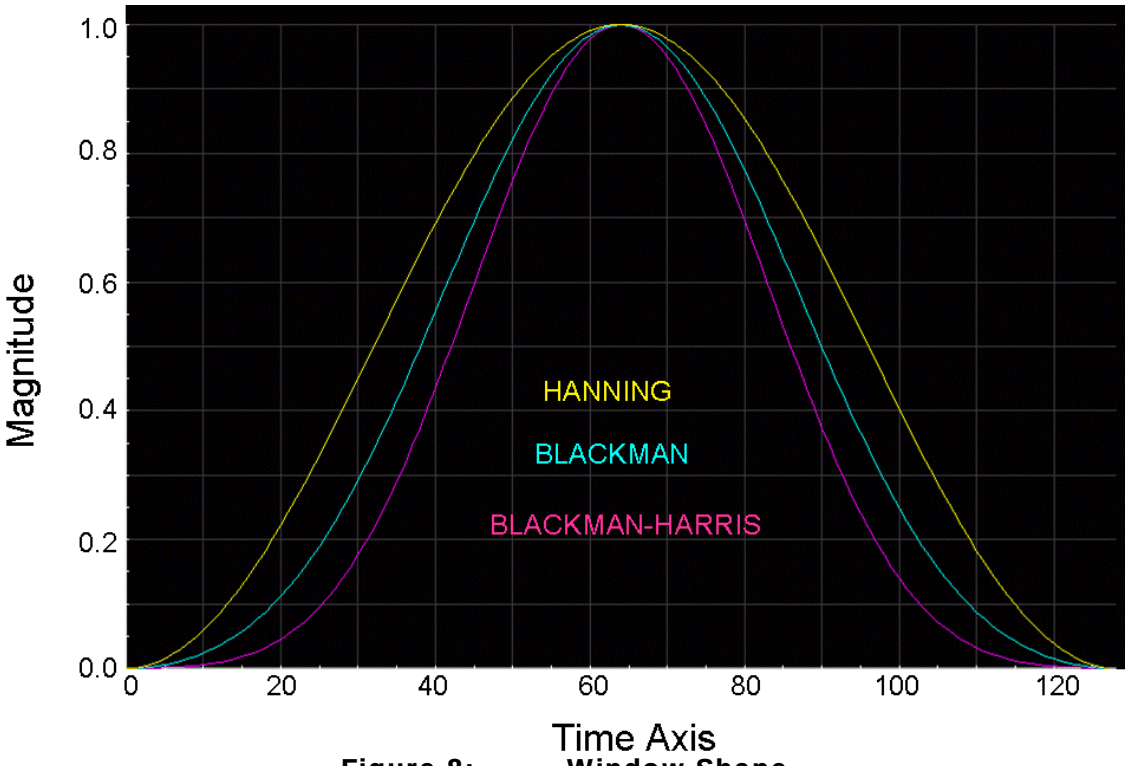

Figure 8: Window Shape

Applying each window to the primitive FIR in Figure 4 and performing the normalization, the weighted FIR's look as Figure 9. You can see both terminals of each FIR converge zero here.

Performing convolution with each weighted FIR, the frequency response of the digital filters is shown in Figure 10. Figure 11 shows the precise gain and phase responses in the pass-band. Among three filters, Blackman-Harris FIR is excellent regarding the pass-band flatness and the rejection. In terms of the phase response in the pass-band, all of them are exactly linear. The linear phase characteristics is very important nature in FIR filters, and it is really useful in wide-band communication applications.

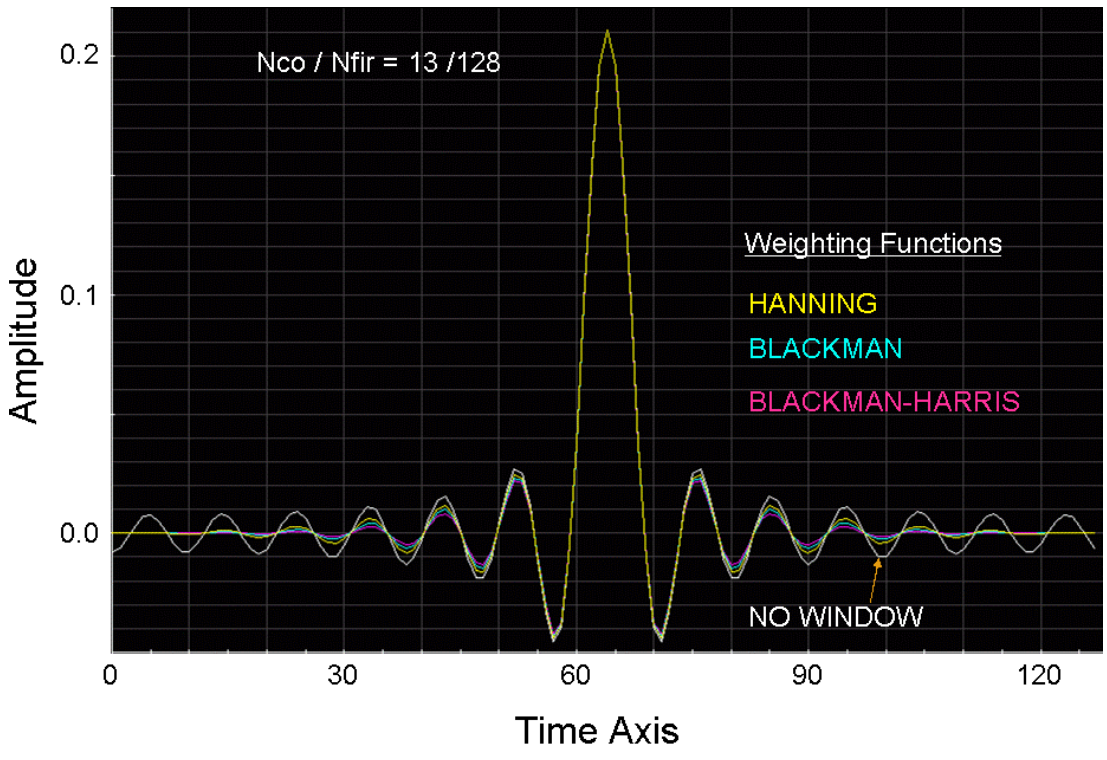

Figure 9: Window Function Weighted FIR

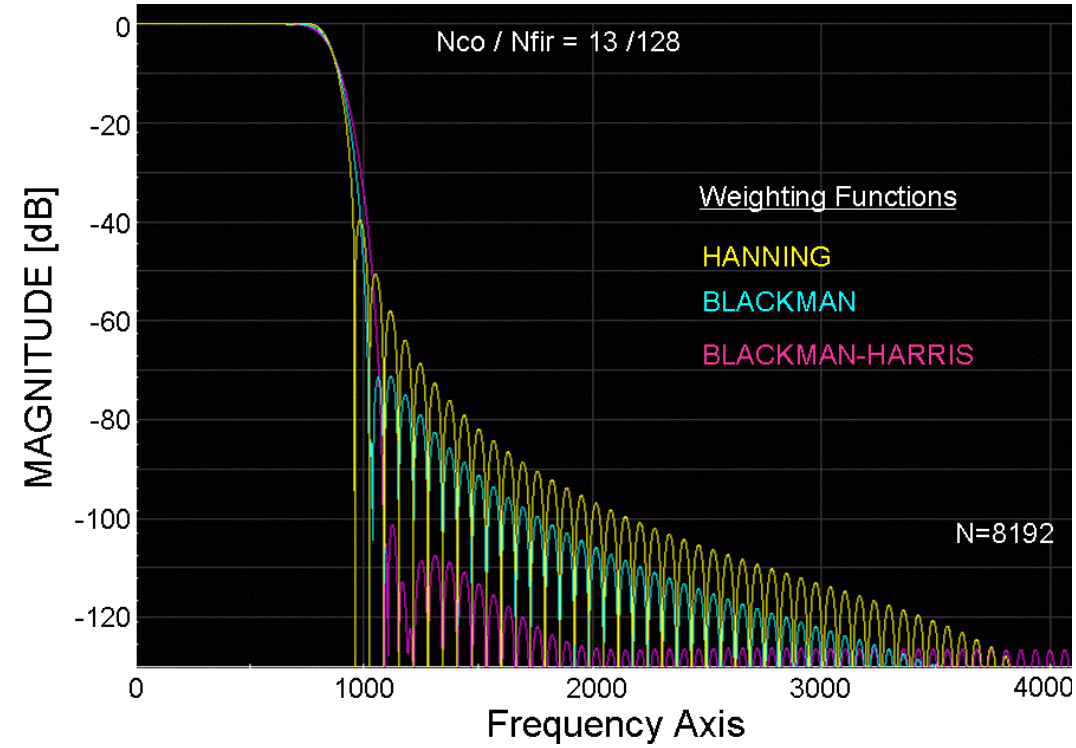

Figure 10: Frequency Responses of Each FIR Filter

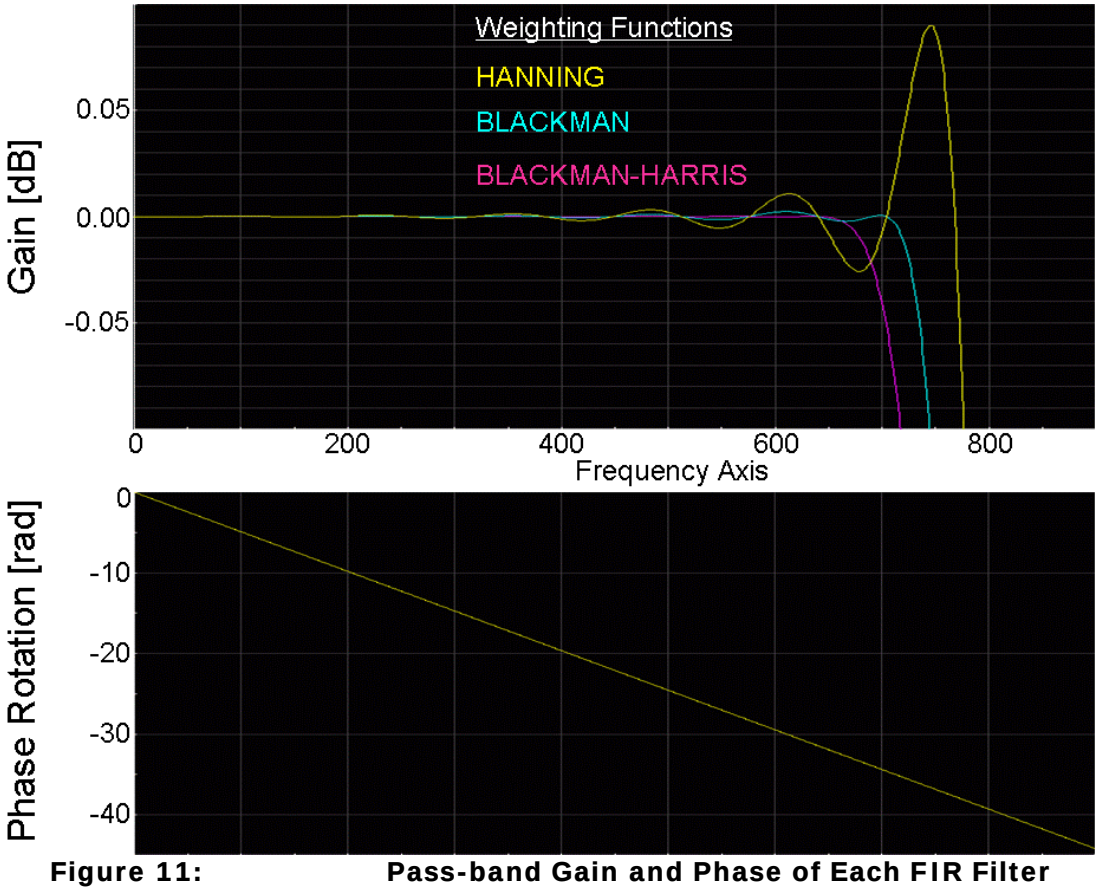

Figure 11: Pass-band Gain and Phase of Each FIR Filter

The three FIR weightings and filter processings are performed as List 7 below.

```
71:     ARRAY_D   dHAN,dFIR1,dFIR_HAN,dWave1;
72:
73:     dHAN.resize(Nfir);                          // Hanning Function
74:     for (i=0;i<Nfir;i++) dHAN[i]=0.5*(1.0-cos(2.0*M_PI*i/Nfir));
75:     DSP_MUL_VEC(dFIR,dHAN,dFIR1);               // Hanning Weighted FIR
76:     Normalize(dFIR1,dFIR_HAN);                  // FIR Normalization
77:     DSP_CONVOL(dFIR_HAN,dVsmp,dWave1);          // Filtering
78:
79:     DOUBLE    dAlpha,dAo,dA1,dA2;
80:     ARRAY_D   dBLK,dFIR2,dFIR_BLK,dWave2;
81:
82:     dBLK.resize(Nfir);                         // Blackman Function
83:     dAlpha=0.16;
84:     dAo=(1.0-dAlpha)/2.0; dA1=0.5; dA2=dAlpha/2.0;
85:     for (i=0;i<Nfir;i++)
86:        dBLK[i]=dAo-dA1*cos(2.0*M_PI*i/Nfir))
87:                   +dA2*cos(4.0*M_PI*i/Nfir));
88:     DSP_MUL_VEC(dFIR,dBLK,dFIR2);              // Blackman Weighted FIR
89:     Normalize(dFIR2,dFIR_BLK);                 // FIR Normalization
90:     DSP_CONVOL(dFIR_BLK,dVsmp,dWave2);         // Filtering
91:
92:     DOUBLE    dA3;
93:     ARRAY_D   dBNH,dFIR3,dFIR_BNH,dWave3;
94:
95:     dBLK.resize(Nfir);                      // Blackman-Harris Function
96:     dAo=0.35875; dA1=0.48829; dA2=0.14128;  dA3=0.01168
97:     for (i=0;i<Nfir;i++)
98:       dBNH[i]=dAo-dA1*cos(2.0*M_PI*i/Nfir))
99:                  +dA2*cos(4.0*M_PI*i/Nfir))
100:                 -dA3*cos(6.0*M_PI*i/Nfir);
101:    DSP_MUL_VEC(dFIR,dBNH,dFIR3);           // Blackman-Harris Weighted
102:    Normalize(dFIR3,dFIR_BNH);              // FIR Normalization
103:    DSP_CONVOL(dFIR_BNH,dVsmp,dWave3);      // Filtering
104:
```

List 7: Practical FIR Filtering

The frequency response in Figure 10 is very improved than the response in Figure 7. This is the effect of window weighting. Figure 10 tells that the roll-off sharpness is incompatible to the rejection. The pass-band flatness is incompatible to the sharpness as well.

6. HPF and BPF

Low pass filters are explained step by step in the previous sections. High pass filters (HPF) and band pass filters (BPF) can be programmed by the same method as well.

Figure 12 shows the planning of a HPF. The cut-off frequency is *Fco* which corresponds to the number of cycles *Nco* in the FIR frequency domain. The constructed FIR looks as Figure 13 and its frequency responses are shown in Figure 14.

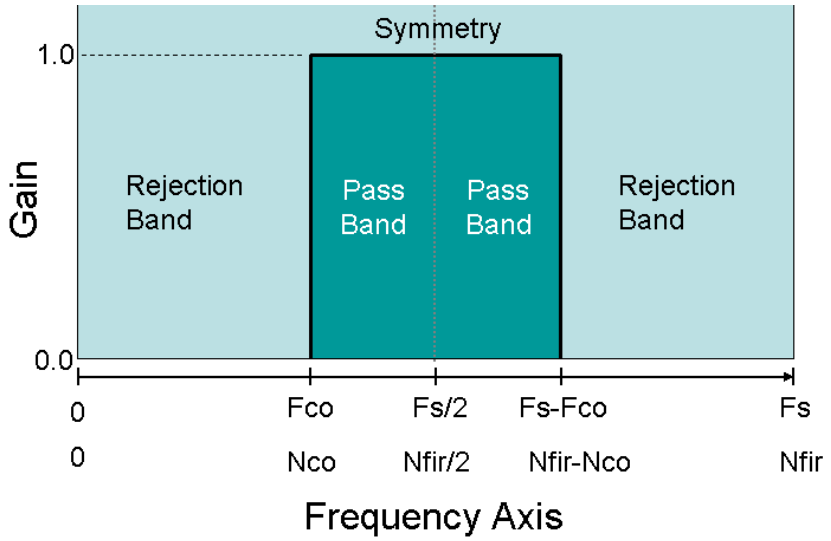

Figure 12: High Pass Filter Planning

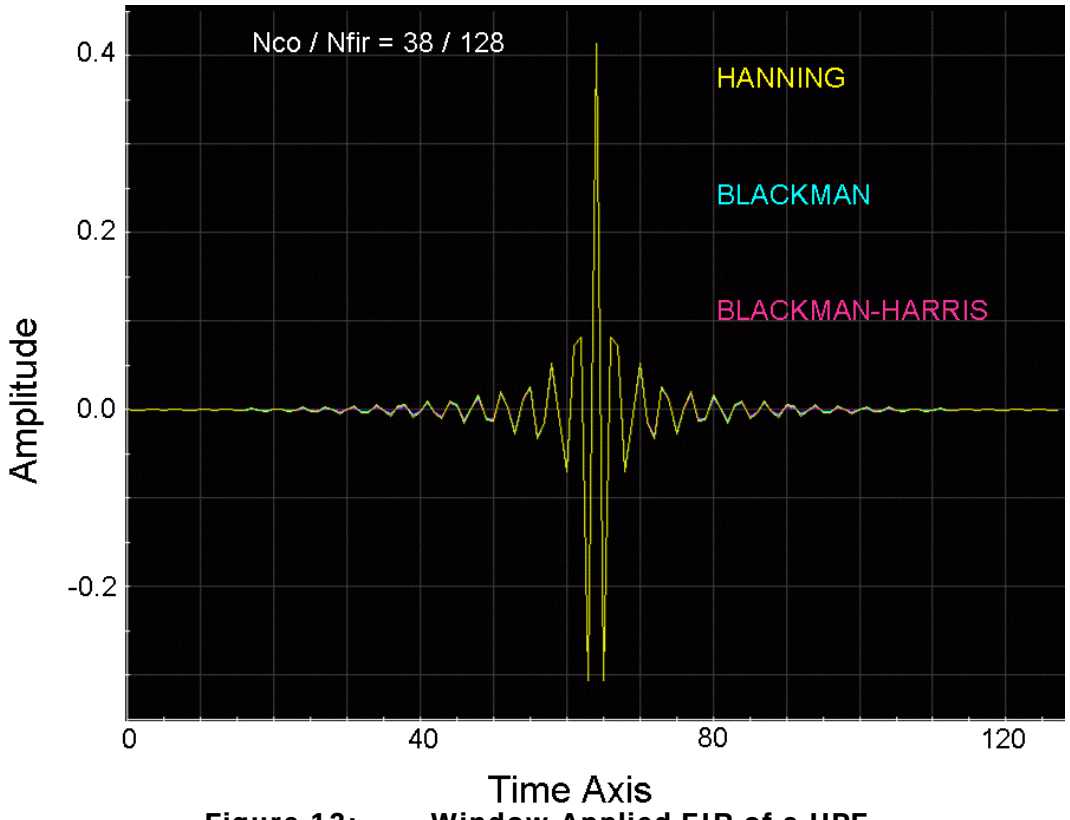

Figure 13: Window Applied FIR of a HPF

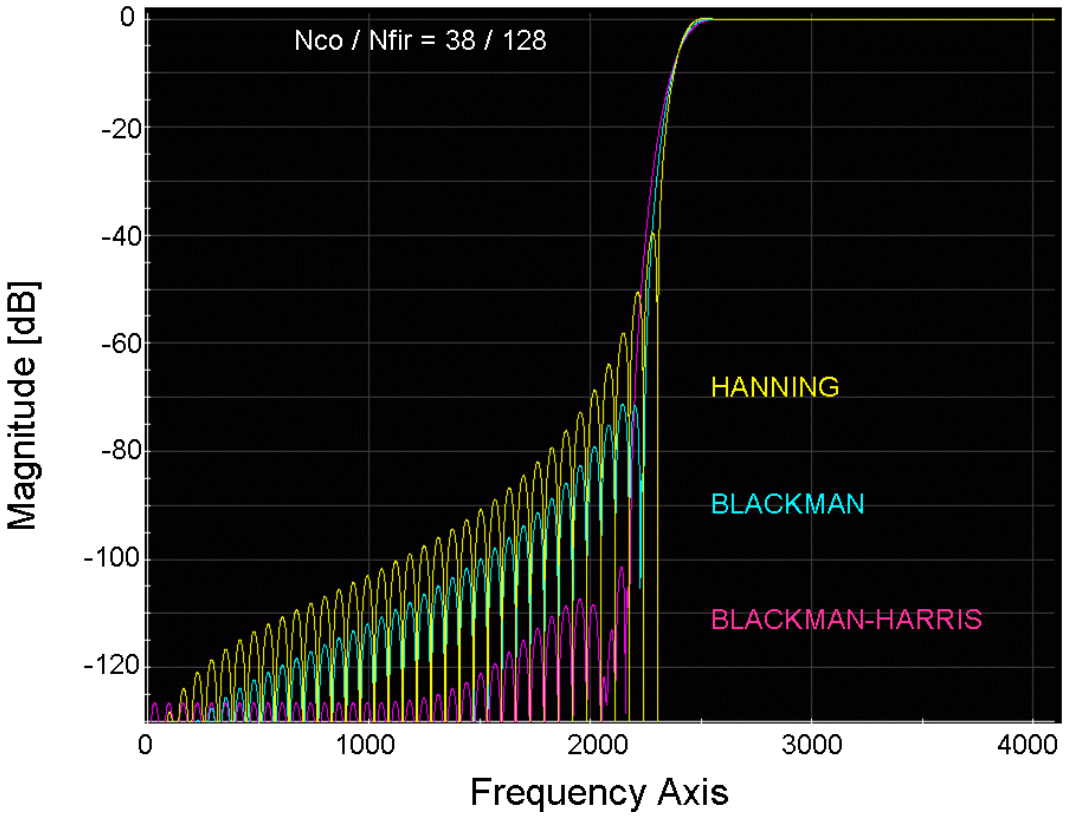

Figure 14: HPF Frequency Response

Figure 15 is a plan of a BPF. The cut-off frequencies are Fco1 and Fco2, which correspond to Nco1 and Noc2 in the FIR domain. The frequency response should be programmed symmetry in the left and the right pages. The BPF plan is converted as the FIR in Figure 16, and the frequency response is realized as Figure 17.

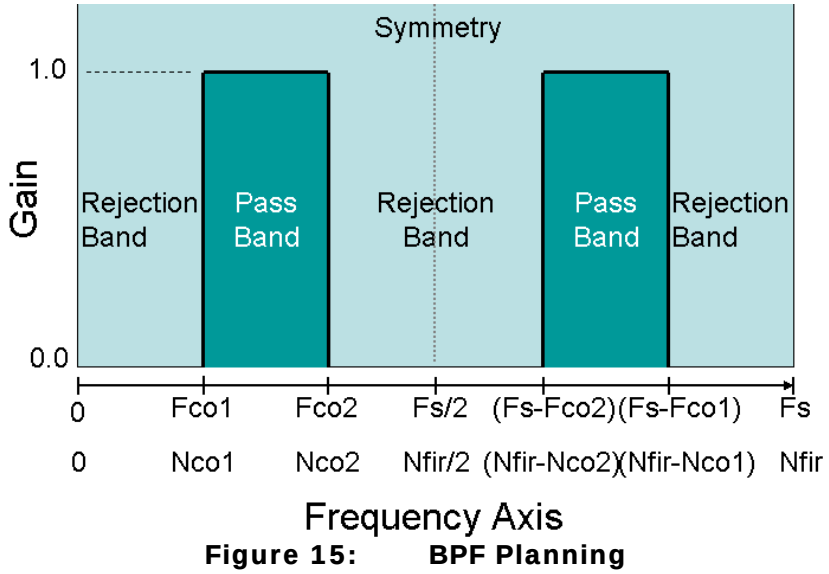

Figure 15: BPF Planning

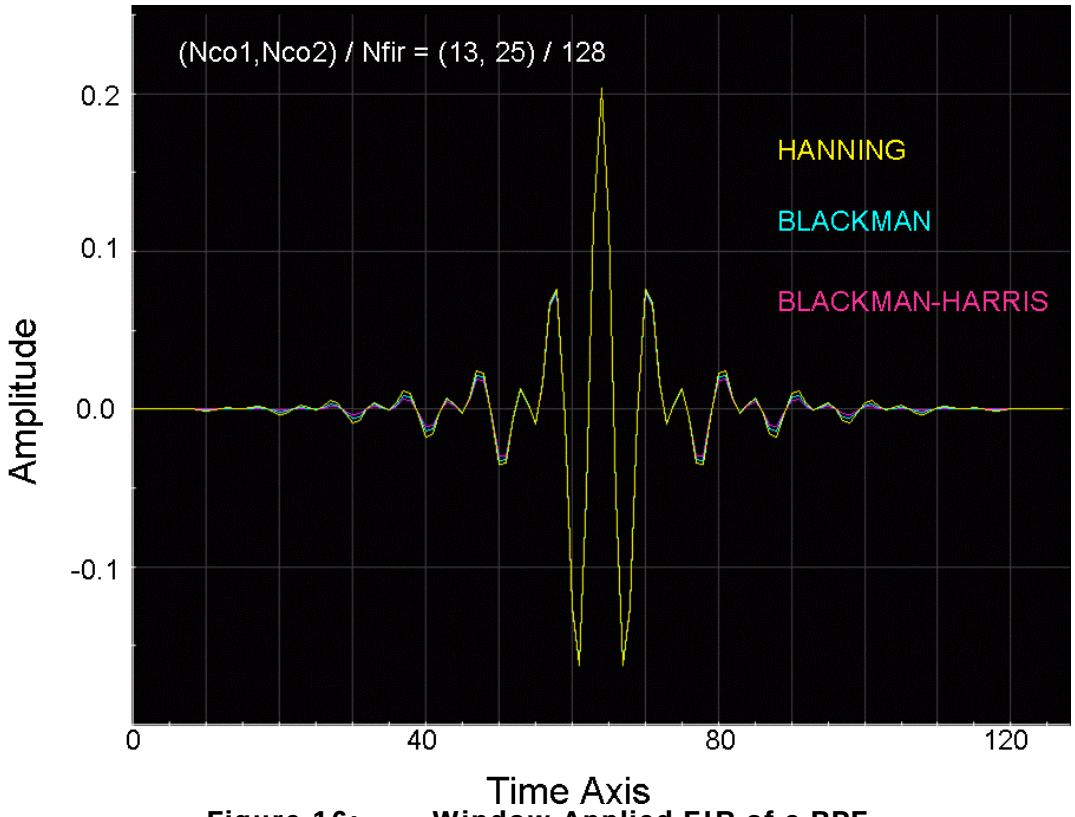

Figure 16: Window Applied FIR of a BPF

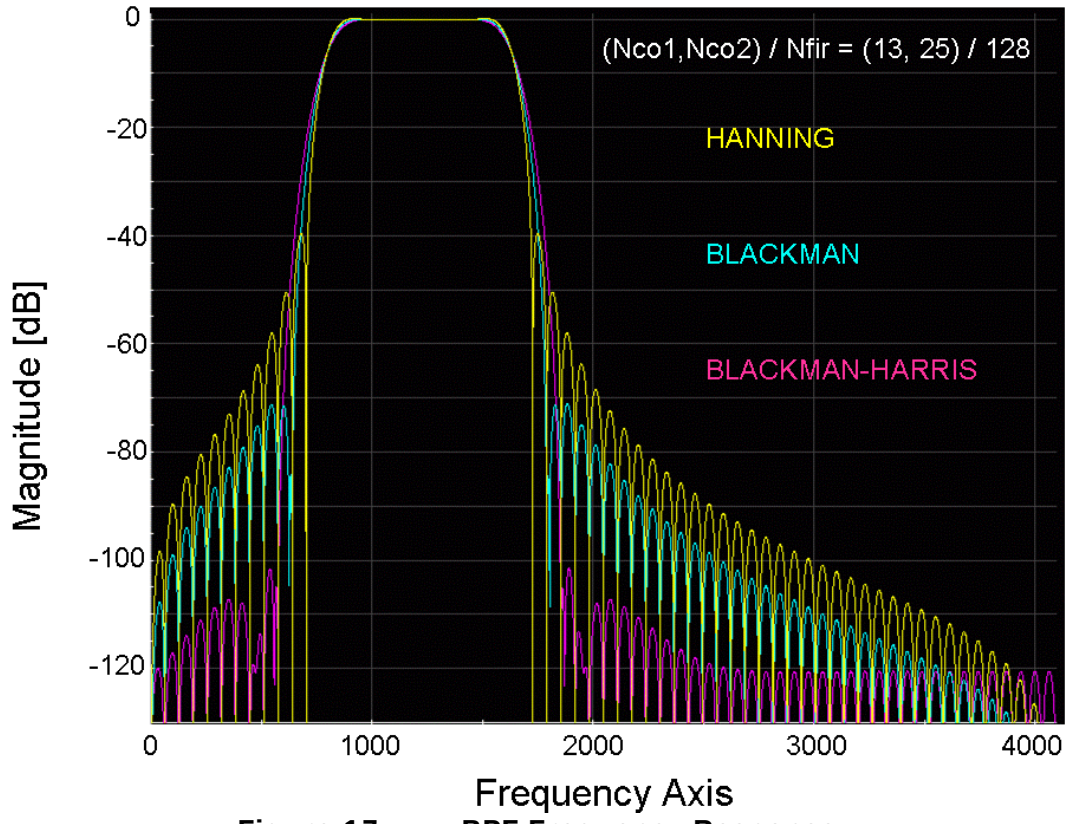

Figure 17: BPF Frequency Response

7.Summary

How to create a FIR for an appropriate filter.

1. Define *Nfir* which should be 2^n.
2. Design the cut-off frequency as *Nco*/*Nfir*, etc.
3. Create a brick-wall frequency plan in the full-page spectrum.
4. Perform IFFT to create a primitive FIR.
5. Reconstruct the split FIR as a normal shape.
6. Apply a weighting window to the FIR.
7. Perform normalization of FIR.
8. Check the actual frequency response.

Figure 18 shows the frequency response vs. the FIR length *Nfir*. The longer the FIR is, the sharper the roll-off becomes. However, convolution is a heavy mathematical operation from the test time point of view so that *Nfir* should be as short as possible for throughput. It should be tuned up during the online debug. Once you fix the shape of the FIR, the array should be stored as constants in your application program for throughput.

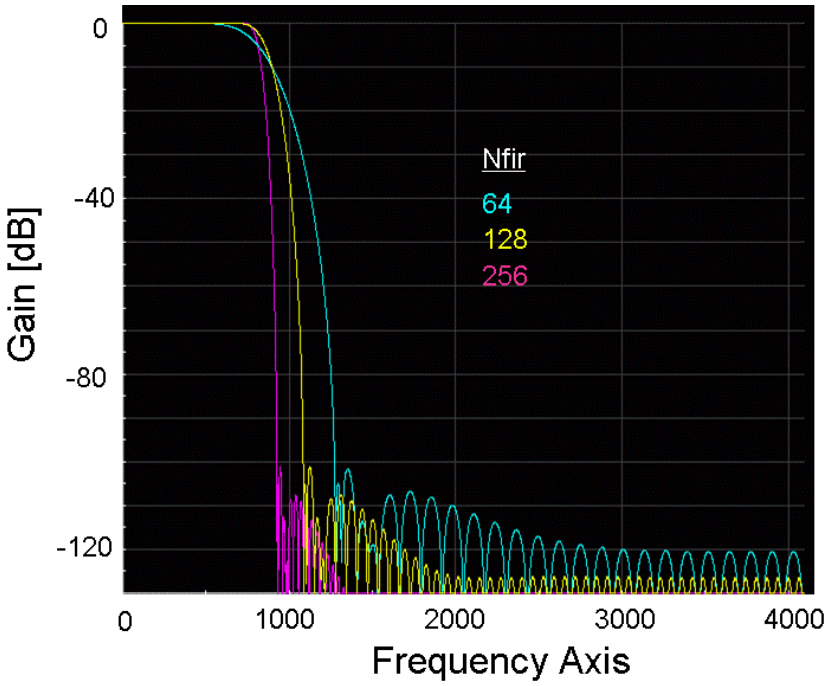

Figure 18: FIR Length Dependency

8. Appendix

Right after performing IFFT to the plan, the primitive impulse response is generated as split image as Figure 3 shows. So you have to exchange the left and right pages in the next step. If you manipulate the phase in the plan, you can create the normal FIR at the first time. See Figure 19(a). The center of the true impulse response is located at the delay of *Nfir*/(2*Fs*). The linear phase trend of the FIR looks as Figure 19(b). The delay of the phase trend can be described as $(\triangle\theta/\triangle\omega) = \triangle\theta/(2\pi Fs/Nfir)$. Therefore if you modify the original frequency plan with adding the phase rotation of $\triangle\theta$, the normal image of the FIR would be expected.

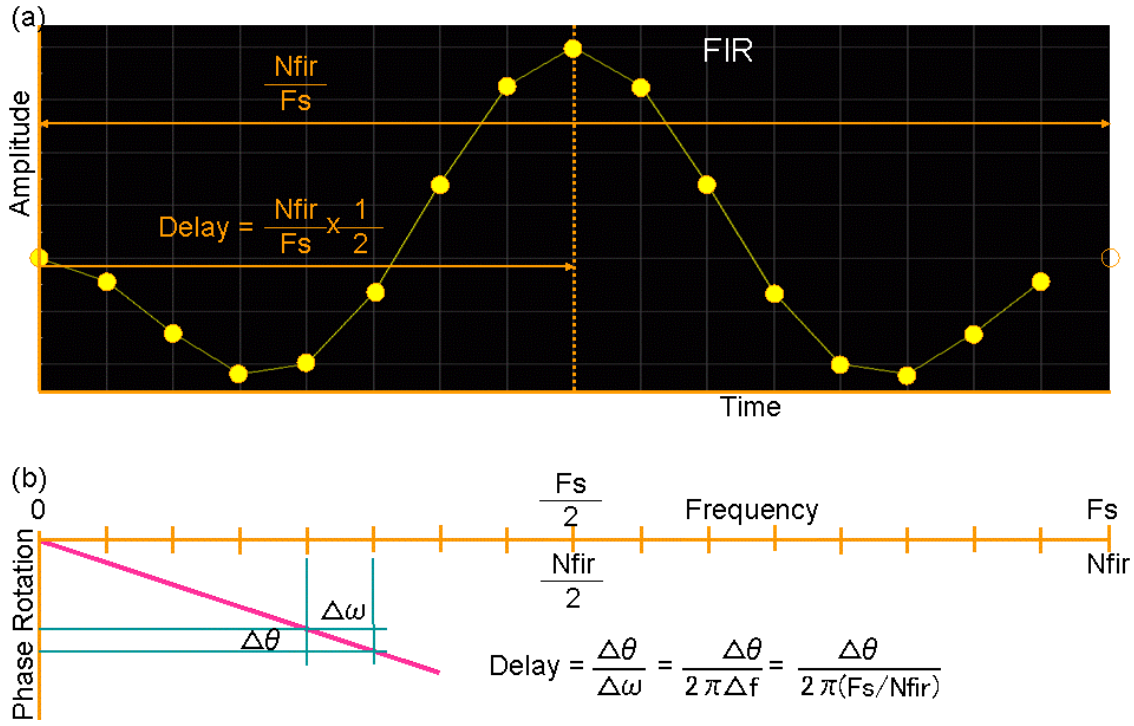

Figure 19: FIR Delay

$$\Delta\theta = 2\pi \frac{Fs}{Nfir} \cdot Delay = 2\pi \frac{Fs}{Nfir} \cdot \frac{Nfir}{2Fs} = \pi \qquad (4)$$

Equation (4) tells that necessary phase rotation is resolved as π at every point.

```
105:    ARRAY_D   dAmp,dPha;
106:
107:    dAmp.resize(Nsp);  dPha.resize(Nsp);
108:    for (i=0;i<=Nco;i++) {
109:       dAmp[i]=1.0;                    // Gain 1.0
110:       dPha[i]=M_PI*i;                 // Phase Rotaion "pi" every step
111:    }
112:    for (i=Nco+1;i<Nsp;i++) {
113:       dAmp[i]=0.0; dPha[i]=0.0;       //  Gain 0.0
114:    }
115:    DSP_POL_RECT(dAmp,dPha,CSp);       //  Polar to Rectangular
116:    CSp.resize(Nfir);                  //  Make CSp full-page
117:    for (i=1;i<=Nsp;i++) {
118:       CSp[Nfir-i].real()= CSp[i].real();  // Complex
119:       CSp[Nfir-i].imag()=-CSp[i].imag();  // Conjugate
120:    }
121:    DSP_IFFT(CSp,CFIR);
122:    dFIR.resize(Nfir);
123:    dFIR=CFIR.getReal();               // Pick up the real part
124:
```

List 8: Modified Program for FIR Generation

Applying the phase rotation of π every step as List 8, the FIR is directly generated as Figure 20. The waveform is already organized.

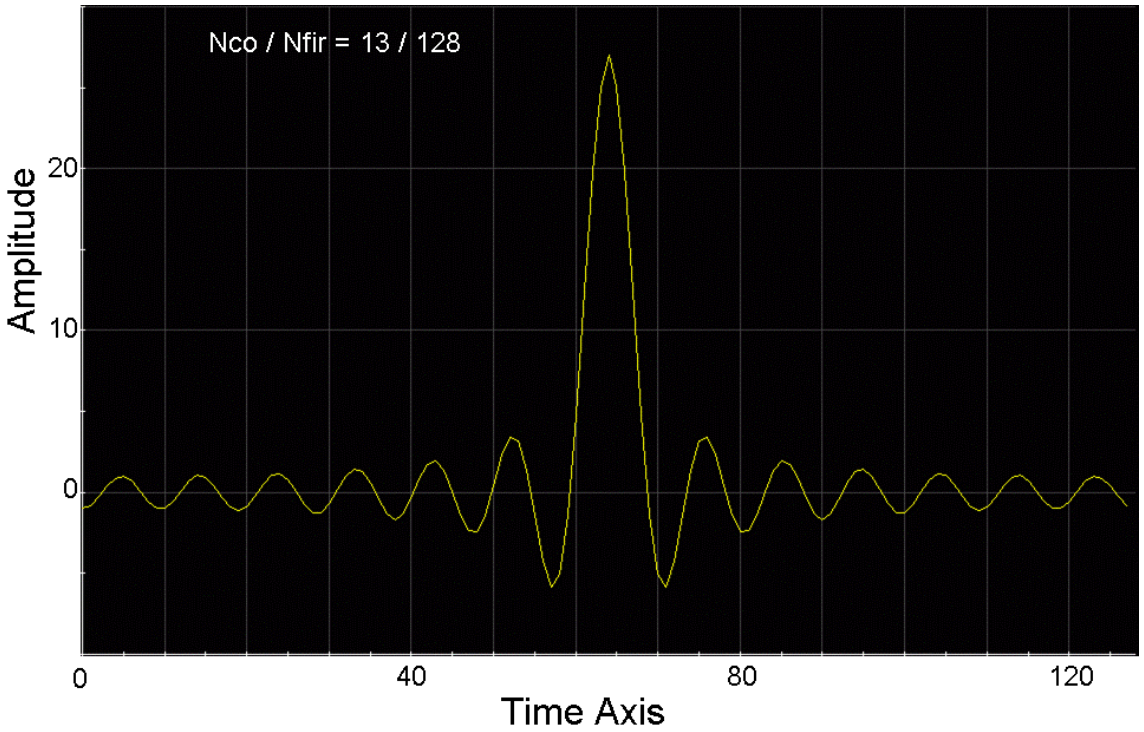

Figure 20: Directly Generated True Image of FIR